Математические структуры и моделирование
2026. № 1 (77). С. 69–77

УДК 004.9 + 528.9
DOI 10.24147/2222-8772.2026.1.69-77

УПРАВЛЕНИЕ СЛОЯМИ ГЕОПРОСТРАНСТВЕННЫХ ДАННЫХ ЧЕРЕЗ СПРАВОЧНИК ТИПОВ СУЩНОСТЕЙ

Т.Е. Болдовская[1]
к.т.н., доцент, e-mail: boldovskaya73@gmail.com
М.И. Ремизов[2]
младший научный сотрудник, e-mail: remizov.maximus.maxim6@gmail.com
А.Е. Ветров[2]
младший научный сотрудник, e-mail: aleksandrvetrv838@gmail.com

[1]Омский государственный технический университет, Омск, Россия

[2]Томский государственный университет, Томск, Россия

Аннотация. Рассматривается архитектура управления слоями геопространственных данных в веб-ориентированной геоинформационной системе, основанная на серверном справочнике типов сущностей. Предлагаемый подход устраняет жёсткую привязку конфигурации слоев к клиентскому коду и обеспечивает синхронизацию структуры отображаемых данных с актуальным состоянием базы данных. Особое внимание уделено серверной кластеризации точечных объектов, адаптивной загрузке данных по текущей области видимости карты и реактивной фильтрации слоев по типам сущностей. Описана многоуровневая схема доступа к данным, включающая OpenAPI-клиент, прикладной API-адаптер и репозиторий, инкапсулирующий агрегацию страниц, преобразование транспортных объектов и обработку ошибок. Показано, что сочетание серверной фильтрации, динамической конфигурации интерфейса и отмены устаревших запросов позволяет повысить производительность, управляемость и безопасность картографического приложения.

Ключевые слова: геоинформационные системы, геопространственные данные, серверная кластеризация, область видимости карты, справочник типов сущностей, динамическая конфигурация слоев, репозиторий.

Введение

Традиционные веб-приложения, работающие с многослойными картографическими представлениями, сталкиваются с рядом архитектурных ограничений при управлении динамическими источниками данных [1, 3]. В типичной архитектуре конфигурация слоев карты жёстко кодируется в исходном коде клиентской части в виде констант или конфигурационных объектов. При добавлении нового слоя разработчик вынужден производить изменения в исходном коде, что ведёт к полному циклу развёртывания (сборка, тестирование, деплой). Помимо этого страдает масштабируемость, поскольку при множественном добавлении слоев каждое изменение требует ручных правок кода.

Когда все слои карты являются постоянными значениями, находящимися на клиентской части приложения, в этом случае они становятся доступными всем пользователям без какой-либо дифференциации: «какие слои видит пользователь» либо полностью отсутствует, либо реализуется через условные конструкции в клиентском коде. Это является серьёзной проблемой безопасности: пользователь может отправить напрямую API-запрос на получение произвольного слоя или изменить переменную в инструментах разработчика в браузере. Поэтому контроль доступа к слоям необходимо выносить на серверную часть приложения, что позволяет отображать только разрешённые действия, логировать все попытки доступа на сервере и исключить возможность обхода проверок через манипуляции с клиентским кодом. При использовании справочника типов сущностей API возвращает только те слои, на которые у пользователя есть права, фильтруя их на основе таблицы слоев и текущей сессии пользователя.

В статье рассматривается комплексное решение, объединяющее серверную кластеризацию, адаптивную загрузку данных по области видимости карты и динамическое управление слоями через справочник типов сущностей.

1. Архитектура управления слоями через метаданные

Практическая реализация управления слоями через метаданные предполагает регистрацию слоя геоданных с указанием названия слоя, описания, проекции, прав доступа и других атрибутов. Геоданные разделяются на слои (классы, темы), при этом для каждого слоя создаётся отдельная директория, что позволяет логически разделить файлы различных слоев и управлять доступом к слоям на уровне директорий [2].

Критически важным является то, что метаданные (данные о данных), описывающие характеристики (содержание, объем, положение в пространстве, качество и т.д.) пространственных данных, содержат сведения о составе, статусе (актуальности и обновляемости), происхождении, местонахождении, качестве, форматах представления и условиях доступа. Это полностью соответствует концепции справочника типов сущностей, где каждый слой карты описывается набором атрибутов: уникальный идентификатор типа, семантическая метка для отображения, текстовое описание, флаг активности и уровень доступа. Применение этих принципов к картографическому контексту позволяет трактовать каждый слой карты как конфигурационный элемент с набором атрибутов, каждый из которых подчиняется формальной метамодели. Централизованное хранилище таких метаданных в БД обеспечивает единственный источник истины (single source of truth) для конфигурации слоев.

Концепция технологии ГИС заключается в создании многослойной электронной карты, опорный слой которой описывает географию территории. При этом критически важным является организация доступа к данным слоев через унифицированный интерфейс, независимый от внутренней структуры хранения. В свою очередь API-контракт действует как формальное соглашение между клиентом и сервером о структуре запросов и ответов, типах данных, ошибочных состояниях и семантике операций.

2. Серверная кластеризация геопространственных данных

Одной из ключевых проблем визуализации пространственных данных на интерактивных картах является необходимость отображения больших объемов точечных объектов при различных масштабах.

Традиционный подход – отправка и отрисовка каждого объекта как индивидуального маркера — неприемлем при масштабах в тысячи и более объектов, поскольку приводит к перекрытию маркеров, утрате читаемости карты и значительному росту нагрузки на сетевой канал и движок рендеринга.

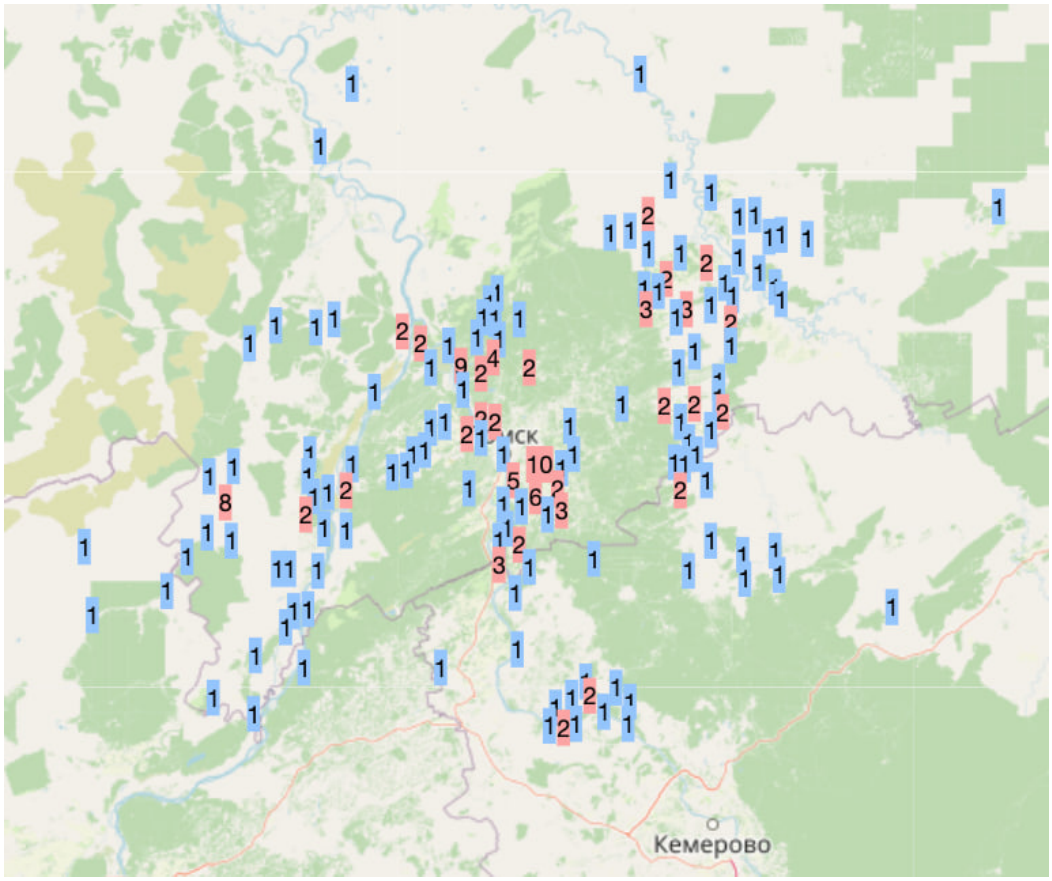

Рис. 1. Обзорное отображение геообъектов на карте

В рассматриваемой архитектуре кластеризация выполняется на серверной стороне, не затрагивая пользовательского интерфейса. Это принципиальное решение обусловлено несколькими факторами. Во-первых, серверная кластеризация позволяет оперировать полным набором данных, включая объекты за пределами текущей области видимости и обеспечивает корректное формирование кластеров на границах видимой области. Во-вторых, результаты кластеризации могут быть кэшированы на сервере для типичных комбинаций масштаба и координат, снижая вычислительную нагрузку при повторных запросах. В-третьих, пользователь получает уже агрегированные данные, что минимизирует объем передаваемого трафика.

Алгоритм серверной кластеризации принимает на вход два ключевых параметра: текущий уровень масштабирования и ограничивающий прямоугольник области видимости. Уровень масштабирования определяет пространственное разрешение кластеризации: при низких уровнях масштаба (обзор больших территорий) радиус кластеризации увеличивается, объединяя больше объектов в единый кластер; при высоких уровнях масштаба (детальный просмотр) радиус уменьшается, постепенно «раскрывая» кластеры до индивидуальных объектов. Ограничивающий прямоугольник задаётся четырьмя координатами: широта и долгота юго-западного угла, широта и долгота северо-восточного угла — что описывает прямоугольную область на земной поверхности, видимую пользователю в данный момент.

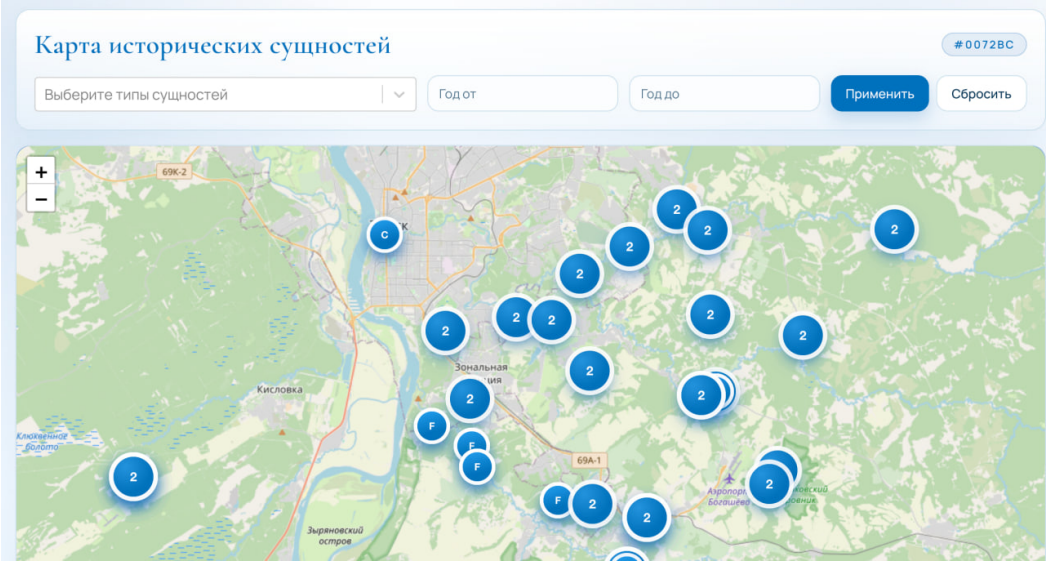

Рис. 2. Результат кластеризации при увеличении масштаба

Результат кластеризации возвращается в формате GeoJSON и содержит два типа геометрических объектов: индивидуальные точки и кластеры. Индивидуальная точка представляет собой единичный геообъект и несёт свойства, позволяющие идентифицировать и описать конкретную сущность. Кластер представляет собой группу пространственно близких объектов и содержит расширенный набор свойств: уникальный идентификатор кластера, количество объединённых объектов, а также коллекцию ссылок на входящие в кластер сущности. Геометрия кластера – это точка, соответствующая центроиду входящих объектов.

Такая дуальная структура ответа позволяет единообразно обрабатывать оба типа данных, визуализируя кластеры как агрегированные маркеры с числовым индикатором количества входящих объектов, а индивидуальные точки – как стандартные маркеры.

Дополнительно API поддерживает фильтрацию по типам сущностей: клиент передаёт массив идентификаторов типов, и сервер выполняет кластеризацию только для объектов, принадлежащих указанным типам. Пользователь получает возможность управлять видимыми «слоями» карты, включая и выключая отображение от-

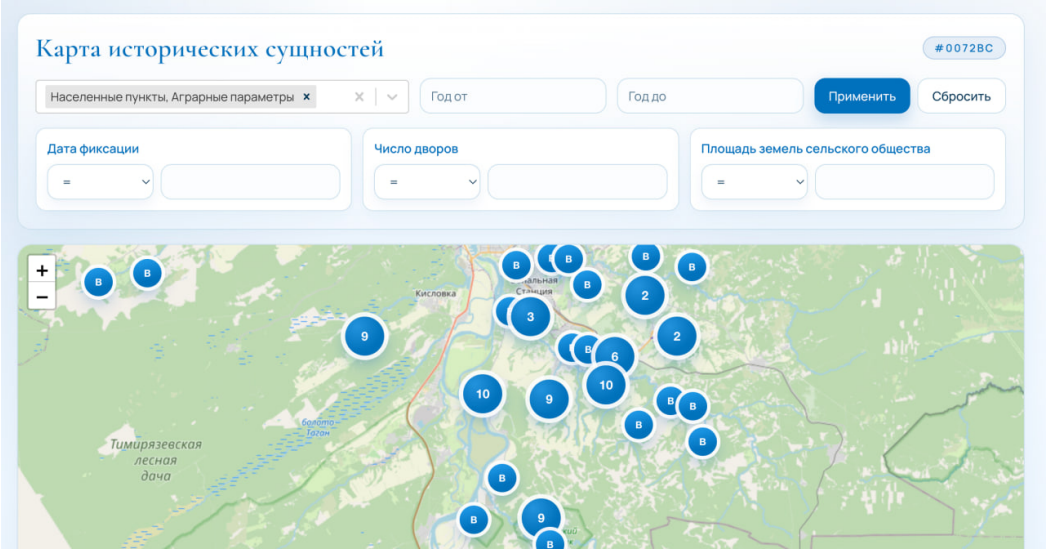

Рис. 3. Результат кластеризации при уменьшении масштаба

дельных классов объектов, при этом кластеризация автоматически пересчитывается с учётом текущего набора активных фильтров.

3. Адаптивная загрузка данных по области видимости

При реализации Информационно-аналитического ресурса «Православный ландшафт таёжной Сибири: акторы, институты, сети» вместо загрузки всего массива геоданных при инициализации карты реализована стратегия загрузки данных по запросу, привязанная к текущей области видимости карты [3]. В качестве примера, показывающего необходимость разработки алгоритмов визуализации объектов разного масштаба, является исследовательская задача по размещению объектов религиозной сети как связанных и коммуницирующих: деревень, имеющих устойчивую географическую локализацию, и таёжных старообрядческих монастырей, местоположение которых могло меняться в силу разных политических, социальных, экономических обстоятельств [4].

Работа алгоритма ресурса осуществляется следующим образом: каждое значимое пользовательское взаимодействие с картой — перемещение, изменение масштаба, поворот – генерирует событие окончания движения. Обработчик этого события извлекает из экземпляра карты актуальные параметры области видимости: центральные координаты (широта и долгота), текущий уровень масштабирования и координаты ограничивающего прямоугольника. Эти параметры сериализуются в строку URL-запроса и сохраняются в адресной строке браузера. Тем самым состояние карты синхронизируется с URL, что даёт два важных преимущества: возможность восстановления точного состояния карты при навигации по истории браузера (кнопки «назад» / «вперёд»), а также возможность передачи ссылки на конкретный вид карты между пользователями.

Обновление параметров в URL запускает реактивный механизм повторного за-

проса данных. Ключ запроса формируется из параметров области видимости и набора активных фильтров, поэтому при изменении любого из этих параметров система выполняет новый запрос к API. Для обеспечения визуальной непрерывности применяется стратегия «сохранения предыдущих данных»: пока новые данные загружаются, на карте продолжают отображаться данные предыдущего запроса, что исключает мерцание и пустую карту в процессе загрузки.

Запрос активируется условно: первоначальный запрос блокируется до момента инициализации карты и определения начальной области видимости. Только после того как карта полностью загружена и сообщила свои координаты через событие onLoad, система выполняет первый запрос данных, таким образом предотвращая отправку запросов с неопределёнными координатами.

Серверный API поддерживает постраничную выдачу данных. Если количество объектов в запрошенной области превышает размер одной страницы, клиент выполняет параллельную загрузку всех оставшихся страниц. Механизм параллельной загрузки осуществляется через систему запросов. Первый запрос возвращает первичные данные (общее количество элементов, количество страниц, текущую страницу и ее размер). На основе этих метаданных формируется массив параллельных запросов для всех оставшихся страниц, которые выполняются одновременно. Результаты всех страниц объединяются: кластеры из всех страниц конкатенируются в единый массив, аналогично объединяются индивидуальные точки. Аналогично механизм объединения применяется и к индивидуальным точкам. В результате чего пользователь получает полный набор данных для текущей области видимости.

Валидация параметров запроса осуществляется на уровне маршрутизации через преобразование, ограничивающее его допустимым диапазоном (между минимальным и максимальным значениями). Координаты центра и ограничивающий прямоугольник являются опциональными параметрами и приобретают значение только после инициализации карты. Набор активных фильтров по типам сущностей также является опциональным, и при отсутствии интерпретируется сервером как «все типы».

4. Репозиторий и многоуровневый доступ к данным

В разработке приложений, требующих отделения бизнес-логики от деталей доступа к данным, широко применяются паттерны Repository и Data Access Object (DAO). Repository представляет собой уровень доступа к данным и выполняет двунаправленную передачу данных между слоями приложения, инкапсулирует логику извлечения данных в отдельном классе, предоставляя чистый, объектно-ориентированный интерфейс для манипуляции доменными сущностями. В рассматриваемой архитектуре доступ к данным организован через трёхуровневую иерархию. На нижнем уровне располагается автоматически сгенерированный API-клиент, полученный из формальной OpenAPI-спецификации серверного контракта. Этот клиент обеспечивает типобезопасное взаимодействие с REST-эндпоинтами и автоматическую сериализацию/десериализацию транспортных объектов (DTO). Средний уровень – прикладной API-адаптер – инкапсулирует сгенерированный клиент, добавляя возможность обработки сигналов отмены (AbortSignal) для прерыва-

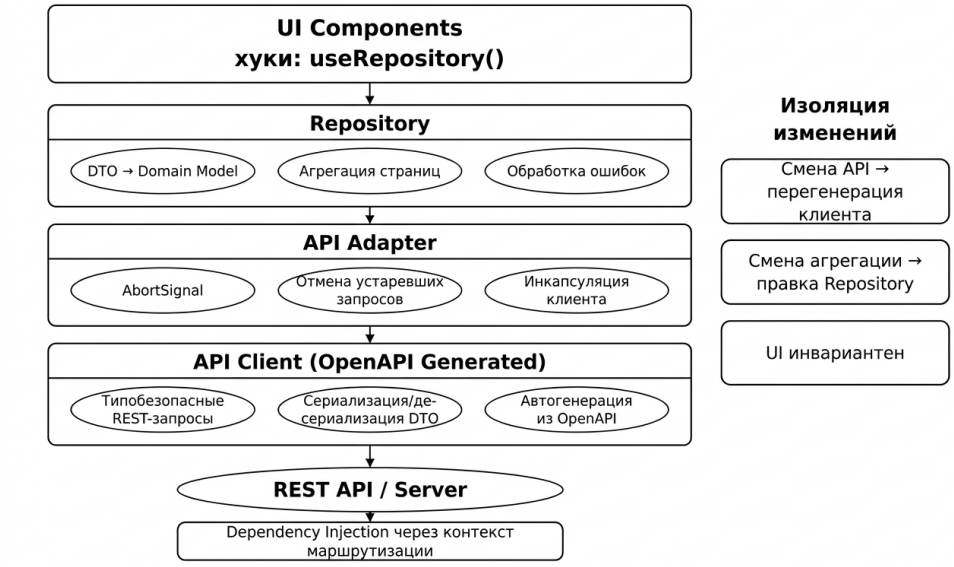

Рис. 4. Репозиторий и многоуровневый доступ к данным

ния устаревших запросов. Верхний уровень – репозиторий – реализует доменную логику: преобразование DTO в доменные модели, агрегацию страниц и обработку ошибок.

Многоуровневая структура, реализуемая в проекте, позволяет изолировать изменения: при обновлении серверного API достаточно перегенерировать нижний уровень; при изменении бизнес-логики агрегации модифицируется только репозиторий. При этом компоненты пользовательского интерфейса остаются инвариантными к обоим видам изменений.

Применительно к справочнику типов сущностей паттерн Repository обеспечивает стабильный интерфейс к справочным данным независимо от выбранной СУБД или стратегии кэширования. Клиентское приложение всегда обращается к актуальному состоянию конфигурации без необходимости глубокого понимания серверной инфраструктуры.

Внедрение зависимостей (Dependency Injection) реализуется через контекст маршрутизации: на каждом уровне иерархии маршрутов создаются экземпляры репозиториев и передаются вниз по дереву компонентов. Компоненты извлекают нужные репозитории с помощью специализированных хуков и обеспечивают инверсию контроля, упрощая тестирование.

5. Динамическая конфигурация слоев и безопасность

В клиентском приложении паттерн динамической конфигурации адаптирует поведение системы в реальном времени с использованием данных, полученных с сервера. При загрузке картографический интерфейс запрашивает у API справочник активных типов сущностей. Сервер возвращает массив объектов с уникальным идентификатором типа, семантической меткой и описанием. Полученные данные интер-

претируются и динамически конструируются в интерфейс управления слоями, позволяющий пользователю включать и отключать отображение определённых классов объектов.

Выбранные пользователем типы сущностей сериализуются в параметры URL наряду с параметрами области видимости карты. При каждом запросе данных на карту массив идентификаторов выбранных типов передаются серверу для выполнения операция фильтрации и кластеризации только объектов соответствующих типов. Данный запрос реализует реактивную связь: изменение набора активных фильтров мгновенно инициирует перезапрос данных с пересчитанной кластеризацией, и карта обновляется без перезагрузки страницы.

При добавлении нового типа сущности на серверной стороне клиентское приложение автоматически отобразит его в интерфейсе фильтрации без необходимости модификации клиентского кода. Это достигается за счёт того, что компонент фильтрации строится полностью на основе данных справочника, а не на основе постоянных значений.

Принцип информационного сокрытия в архитектуре приложений предполагает, что клиент должен иметь доступ только к той информации, необходимой для выполнения его непосредственных задач.

Информационная система ресурса содержит метаданные о слоях(название, описание, проекция, права доступа). Доступ к конкретным слоям регламентируется через систему прав доступа, исключая возможность несанкционированного получения данных. Система управляет доступом к слоям на уровне серверной фильтрации, обеспечивая изоляцию данных. Такая абстракция значительно сокращает поверхность атаки. Невозможно провести SQL-инъекцию или reconnaissance-атаку, основанную на методе перебора имён таблиц.

Дополнительным аспектом безопасности является механизм отмены устаревших запросов. При быстром изменении области видимости карты (последовательные операции масштабирования или панорамирования) предыдущие, ещё не завершённые запросы автоматически отменяются через сигнал отмены (AbortSignal), что снижает нагрузку на сервер.

Заключение

Рассмотренная архитектура демонстрирует комплексный подход к управлению геопространственными данными в интерактивных картографических системах. Серверная кластеризация с параметрическим управлением через уровень масштабирования и ограничивающий прямоугольник обеспечивает эффективную визуализацию больших объёмов данных при любом масштабе просмотра. Адаптивная загрузка данных по области видимости с механизмом параллельной постраничной загрузки, сохранением предыдущих данных и условной активацией запросов обеспечивает отзывчивый пользовательский интерфейс без избыточного сетевого трафика. Динамическое управление слоями через серверный справочник типов сущностей с полной синхронизацией фильтров и параметров карты через URL обеспечивает расширяемость системы без необходимости модификации клиентского кода. Совокупность этих подходов формирует архитектурный паттерн, применимый к

широкому классу геоинформационных систем, оперирующих динамическими, многослойными и объёмными наборами пространственных данных.

Благодарности

Данная работа выполнена при поддержке гранта Российского научного фонда № 23-78-10119.

Литература

1. Кухаренко Е.Л. Об опыте создания сервиса визуализации геоданных без программирования и его использовании в муниципальном образовании // Вестник СГУГиТ. 2025. Т. 30, № 5. С. 41–57.
2. Болдовская Т.Е., Ремизов М.И., Ветров А.Е. Систематизация и сохранение исторических источников в условиях цифровой эпохи // Математические структуры и моделирование. 2025. № 2 (74). С. 90–97.
3. Болдовская Т.Е., Гресь В.И. Информационно-аналитический ресурс «Православный ландшафт таежной Сибири: акторы, институты, сети»: архитектура системы, ключевые характеристики и интеграция исторических геоданных // Историческая информатика. 2025. № 1. С. 73–82. URL: `https://nbpublish.com/library_read_article.php?id=74030` (дата обращения: 10.03.2026).
4. Дутчак Е., Ким Е., Буркун А. Крестьянская община и старообрядческий скит: формула притяжения // Православная культура вчера и сегодня / науч. ред. Е. Потехина, А. Кравецкий. Olsztyn: Uniwersytetu Warminsko-Mazurskiego w Olsztyni, 2015. С. 235–251.

MANAGING GEOSPATIAL DATA LAYERS VIA AN ENTITY TYPE DIRECTORY

T.E. Boldovskaya[1]
Ph.D. (Techn.), Associate Professor, e-mail: boldovskaya73@gmail.com
M.I. Remizov[2]
Junior Scientist Researcher, e-mail: remizov.maximus.maxim6@gmail.com
A.E. Vetrov[2]
Junior Scientist Researcher, e-mail: aleksandrvetrv838@gmail.com

[1]Omsk State Technical University, Omsk, Russia
[2]Tomsk State University, Tomsk, Russia

Abstract. The paper considers an architecture for managing geospatial data layers in a web-based geographic information system using a server-side entity type directory. The proposed approach removes hard-coded layer definitions from the client and synchronizes the displayed configuration with the current database state. Special attention is paid to server-side clustering of point objects, viewport-adaptive data loading, and reactive filtering by entity types. A multilayer data access scheme is described, including an OpenAPI client, an application API adapter, and a repository responsible for page aggregation, DTO-to-domain transformation, and error handling. The results show that combining server-side filtering, dynamic UI configuration, and cancellation of obsolete requests improves performance, maintainability, and security of the cartographic application.

Keywords: geographic information systems, geospatial data, server-side clustering, map viewport, entity type directory, dynamic layer configuration, repository.

Дата поступления в редакцию: 10.03.2026